Amendments to the Specification:

Please replace paragraph [0005] with the following amended paragraph:

[0005] Evolution diagrams can be used to graphically display operation of a distributed or concurrent software system to a programmer to aid in debugging. A preferred diagram explicitly shows selected events, message traffic between software components, changes in component behavior, and correlations between local behavior changes. In one embodiment of such a display, each selected component is represented by a corresponding trace, interface states are displayed by corresponding traces (for example, extending generally parallel to the component traces), and events are made explicit by graphical symbols spanning the space affected by the event. Messages, represented by arrows, present explicit dependencies, while indirect or implicit dependencies such as those implemented by coordinator constraints and actions, are indicated by diagonal lines or the like between the relevant events. ~~See page 78.~~

Please replace paragraph [0011] with the following amended paragraph:

[0011] Additional aspects and advantages ~~of this invention~~ will be apparent from the following detailed description ~~of preferred embodiments thereof~~, which proceeds with reference to the accompanying drawings.

Please replace paragraph [0090] with the following amended paragraph:

[0090] FIG. 2 is component 100 further including a first coordination interface 200, a second coordination interface 202, and a third coordination interface ~~204~~ 206. Coordination-centric design's components 100 provide the code-sharing capability of object-oriented inheritance through copying. Another aspect of object-oriented inheritance is polymorphism through shared interfaces. In object-oriented

languages, an object's interface is defined by its methods. Although coordination-centric design's actions 104 are similar to methods in object-oriented languages, they do not define the interface for component 100. Components interact through explicit and separate coordination interfaces, in this figure coordination interfaces 200, 202, and ~~204~~ 206. The shape of coordination interfaces 200, 202, and ~~204~~ 206 determines the ways in which component 100 may be connected within a software system. The way coordination interfaces 200, 202, and ~~204~~ 206 are connected to modes 102 and actions 104 within component 100 determines how the behavior of component 100 can be managed within a system. Systemwide behavior is managed through coordinators (see FIG. 4B and subsequent).

Please replace paragraph [0098] with the following amended paragraph:

[0098] The coordination-centric design methodology provides an encapsulating formalism for coordination. Components such as component 100 interact using coordination interfaces, such as first, second, and third coordination interfaces 200, 202, and ~~204~~ 206, respectively. Coordination interfaces preserve component modularity while exposing any parts of a component that participate in coordination. This technique of connecting components provides polymorphism in a similar fashion to subtyping in object-oriented languages.

Please replace paragraph [0490] with the following amended paragraph:

[0490] FIG. 43 portrays the evolution of a dedicated RPC transaction. It has traces for both components ~~4310~~ (bars enclosed by dashed lines), interface states ~~4312~~ (shown by solid horizontal bars), and events ~~4314~~ (shown by vertical ovals spanning the space affected by the event). The figure displays all essential aspects of an RPC transaction.